ECS 253: Computer Security — Spring Quarter–1997
Scribe: John Black — 05/05/97

ECS 253 Lecture, May 5th, 1997

Basis of Security

Safety Question

Given an abstract system, can we determine if it's safe? That is, given an arbitrary system with an arbitrary set of rights, can we determine if the system is safe?

ACM Model

We will use the ACM (Access Control Model) as a model for our system's security. We have only two entities: subjects and objects. Subjects act upon things, and objects are acted upon; but are subjects also objects? In other words, can subjects be acted upon? The answer is "it depends." For the first model we're going to talk about, a subject is also an object.

Security depends on "rights." Commonly these are read, write, append, execute, or own (where own means the right to change rights, typically).

Here is a matrix indicating which entities have which rights:

	o_1	o_2	o_3	...	p_0	p_1	p_2
p_0	r	rw	rwx	...	w	o	o
p_1	w	a		...	r	r	r
p_2	x	x	rx	...	r	x	w

In the above table, the p_i are people (subjects) and the o_i are objects. The matrix is meant to represent which subjects have what rights over what other subjects and objects. For example, p_0 has read and write rights over o_2. Each column is like a Unix "Access Control List."

Operations on the Access Control Matrix

We are going to define several actions to manipulate the matrix given above; the notation is formal, set-theoretic stuff, but the ideas are quite natural.

Let $r \in R$ be a right. Let A be the ACM matrix and $A[p, o]$ be the entry of A at row p and column o. Let S be the set of subjects, and O be the set of objects. We define the following operations on the ACM:

1. Enter right r into $A[p, o]$ **before:** (S, O, A) **after:** (S', O', A') where the primed sets are determined as follows: $S' = S$ and $O' = O$. In other words, no new subjects or objects are created as a result of this command (which follows from our intuition: we are adding a right into the matrix). Now how do we determine A', the new version of the matrix. Well, we just go down each row; if the current subject p' is the subject p we are looking for, and the current object o' is the object o we're looking for, then we update $A'[p', o']$ to contain the new right r.

If not, we just let $A'[p', o']$ stay the same. Here is it in math notation:

$$\begin{array}{rl} \text{if } p' = p,\ o' = o & A'[p', o'] = A[p, o] \cup \{r\} \\ \text{if } p' \neq p \vee o' \neq o & A'[p', o'] = A[p, o] \end{array}$$

From here on, we'll just present the terse mathematical notation instead of explaining it.

2. Delete right r from $A[p, o]$

$$\begin{array}{rl} S' = S, & O' = O \\ \text{if } p' = p,\ o' = o & A'[p', o'] = A[p, o] - \{r\} \\ \text{if } p' \neq p \vee o' \neq o & A'[p', o'] = A[p, o] \end{array}$$

3. Create Subject p

$$\begin{array}{rl} S' = S \cup \{p\}, & O' = O \cup \{p\} \\ \text{if } p \in S \wedge o \in O & A'[p', o'] = A[p, o] \\ \text{if } p' = p \vee o' = o & A'[p', o'] = \emptyset \end{array}$$

4. Create Object o

$$\begin{array}{rl} S' = S \cup \{o\}, & O' = O \cup \{o\} \\ \text{if } o \in O & A'[p', o'] = A[p, o] \\ \text{if } o' = o & A'[p', o'] = \emptyset \end{array}$$

5. Destroy Subject p

$$\begin{array}{rl} S' = S - \{p\}, & O' = O - \{p\} \\ \text{if } p \neq p \wedge o' \neq o & A'[p', o'] = A[p, o] \end{array}$$

6. Destroy Object o

$$\begin{array}{rl} S' = S - \{o\}, & O' = O - \{o\} \\ \text{if } o' \neq o & A'[p', o'] = A[p, o] \end{array}$$

Higher level commands bulit on above primitives

The primitives defined above are too low-level for our purposes, so we will create higher-level commands which use the preceding as subroutines. The format of these commands will be

$$\text{command}(p_1, \ldots, p_n, o_1, \ldots, o_n)$$

and the implementation of this will look like this:

$$
\begin{array}{rl}
\text{if} & r_1 \in A[p_1, o_1] \wedge r_2 \in A[p_2, o_2] \wedge \ldots \wedge r_n \in A[p_n, o_n] \\
\text{then} & \\
 & op_1 \\
 & op_2 \\
 & \vdots \\
 & op_n \\
\text{end} &
\end{array}
$$

Let's do an example. Let's say you want to create a file, give the ownership right, and read-write access. The command would be:

$$cf(p, f)$$

where cf means create file, and p is the subject, and f is the file (the object). The steps are as follows: create object f, enter *own* into the matrix at $A[p, f]$, enter r (the read permission), into $A[p, f]$ and enter w (the write permission), into $A[p, f]$. End.

Notice this didn't use any conditionals. Let's do another example, with a condition. Let's do "grant_read". This means we want to let p allow q (another subject) to read f. So the command is:

$$\text{grant\_read}(p, q, f)$$

And the algorithm is as follows:

$$
\begin{array}{rl}
\text{if} & own \in A[p, f] \\
\text{then} & \\
 & \text{enter } r \text{ into } A[q, f] \\
\text{end} &
\end{array}
$$

Notice that we had to check to make sure p owned f before it could hand out read rights to q. Note that not all models allow a subject to give away a right to another subject. Some models allow it and some don't. However, never can we allow a subject to give away a right it does not have. This would violate the *Principle of Attenuation* which states that the holder of a right may be able to share, but can't give away rights he/she does not own. (Exception: if you own the file, you can still give away rights on the file.)

Definitions of some terms

Let's define a few terms. If the number of conditions in the preceding algorithms is 1, then we call it "mono-conditional"; if there is only one operation, we call it "mono-operational".

We're now (finally!) ready to define what "safety" is! A system is a set of states; as the matrix changes, so does the state of the system. A state is "unauthorized" iff a command c run in state Q executes a primitive operation entering a right r into a cell of A did not have r initially.

Theorem. There exists an algorithm deciding whether for a given mono-operational system and initial state Q_0 is safe for a given generic right r.

Proof. Look at all operations. If removal operation (delete or destroy) then we have no worries since this just results in things going away in the matrix. Create can also be ignored because no rights are added (except for initial create subject if no initial subject). So "enter" is the crucial operations we must analyze. Let g = number of generic rights. Let $n_s = |S| + 1$ and $n_O = |O| + 1$. We have an upper bound on the number of states of

$$2^{g n_O n_S + 1}$$

That is to say, the number of states is the number of rights, times the number of objects times the number of subjects, plus one for the initial create. Each of these can be off or on, so we have 2 to this power possible states of the matrix. Since the number of states is finite, we just have an algorithm which evaluates all the possible states.

This problem is in NP, but it *is* decidable.

Without the mono-operational constraint, we get an undecidable problem, as we see in the next section.

Harrison-Ruzzo-Ullman (HRU)

This important theorem was stated, but not proven; this will be taken up next lecture. Here is the theorem statement:

Theorem. It is undecidable whether a given state of a given protection system is safe for a given generic right.

The proof of this maps the matrix into a Turing machine and reduces the question of security to the halting problem. Since the halting problem is undecidable, then so is the security problem above. The proof will be given in the next set of notes.